CISC - Complex Instruction Set Computer

MAC 344- Arquitetura de Computadores

- Siang W. Song

Baseado no livro de Tanenbaum - Structured Computer Organization

Índice

Como surgiu a micropramação

Maurice Wilkes foi um cientista da computação britânico que construiu o computador EDSAC, sucessor do ENIAC. EDSAC foi um dos primeiros computadores a armazenar o programa na memória do computador. Em 1951 inventou a microprogramação que revolucionou o projeto de processadores. Wilkes foi professor emérito da Universidade de Cambridge. Em 1967 recebeu o Turing Award.

Como surgiu a micropramação

- Os primeiros computadores tinham poucas instruções, todas implementadas em hardware.
- Maurice Wilkes (1951) introduziu a microprogramação, que permite
  - um conjunto grande de instruções de máquina (no nível convencional), usando um hardware simples capaz de executar apenas as chamadas microinstruções.

Exemplos de máquinas CISC

- Cada instrução de máquina no nível convencional é interpretada e pode dar origem à execução de muitas microinstruções.
- Os computadores que usam microprogramação são ditas CISC - Complex Instruction Set Computer.
- Exemplos: IBM 360, DEC VAX, Motorola 68030, família Intel como 8088, 80386, Pentium etc.
- Tanenbaum usa uma máquina fictícia chamada MAC para ilustrar o conceito de microprogramação.

A máquina MAC

No nível convencional a máquina MAC apresenta as características:

- Memória com 4096 palavras de 16 bits (endereço 12 bits)
- Processador com 16 registradores, incluindo:
  PC (program counter)
  AC (acumulador)
  SP (stack pointer)
- Instrução de máquina (nível convencional) de 16 bits.

Conjunto de instruções da máquina MAC

0000xxxxxxxxxxx LODD ac:=m[x]
0001xxxxxxxxxxx STOD m[x]:=ac
0010xxxxxxxxxxx ADDD ac:=ac+m[x]
0011xxxxxxxxxxx SUBD ac:=ac-m[x]
0100xxxxxxxxxxx JPOS if ac >= 0 then pc:=x
0101xxxxxxxxxxx JZZR if ac=0 tehn pc:=x
0110xxxxxxxxxxx JUMP pc:=x
0111xxxxxxxxxxx LOCO ac:x (0 <= x <= 4095)
1000xxxxxxxxxxx LODL ac:=m[sp+x]
1001xxxxxxxxxxx STOL m[x+sp]:=ac
1010xxxxxxxxxxx ADDL ac:=ac+m[sp+x]
1011xxxxxxxxxxx SUBL ac:=ac-m[sp+x]
1100xxxxxxxxxxx JNEG if ac<0 then pc:=x
1101xxxxxxxxxxx JNZE if ac not= 0 then pc:=x
1110xxxxxxxxxxx CALL sp:=sp-1; m[sp]:=pc; pc:=x
1111000000000000 PSHI sp:=sp-1; m[sp]:=m[ac]
1111001000000000 POPI m[ac]:=m[sp]; sp:=sp+1
1111010000000000 PUSH sp:=sp-1; m[sp]:=ac
1111011000000000 POP ac:=m[sp]; sp:=sp+1
1111100000000000 RETN pc:=m[sp]; sp:=sp+1
1111101000000000 SWAP tmp:=ac; ac:sp; sp:tmp
11111100yyyyyyyy INSP sp:=sp+y (0 <= y <= 255)

11111110yyyyyyyy DESP sp:=sp-y (0 <= y <= 255)

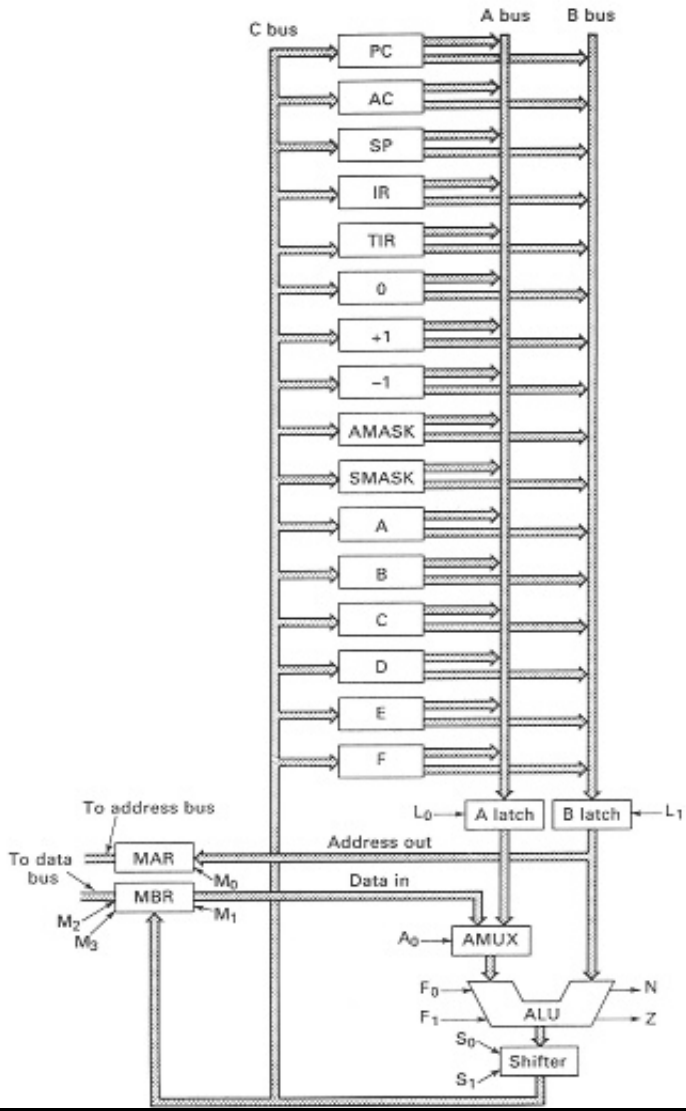
Não precisam decorar isso :-)
Apenas notem que esse conjunto pode ser bem grande e complexo.

A arquitetura MIC

A arquitetura do processador (chamada MIC) é simples e não implementa as instruções de máquina do nível convencional diretamente.
Apresenta os seguintes componentes:

- Uma ALU capaz de fazer apenas 4 operações simples
- Um shifter para deslocar 1 bit para direita ou para esquerda
- 16 registradores
- Um multiplexador MUX de duas entradas
- Três decodificadores 4-para-16
- Registradores MAR e MBR servindo de interface com a memória
- Um relógio de 4 fases

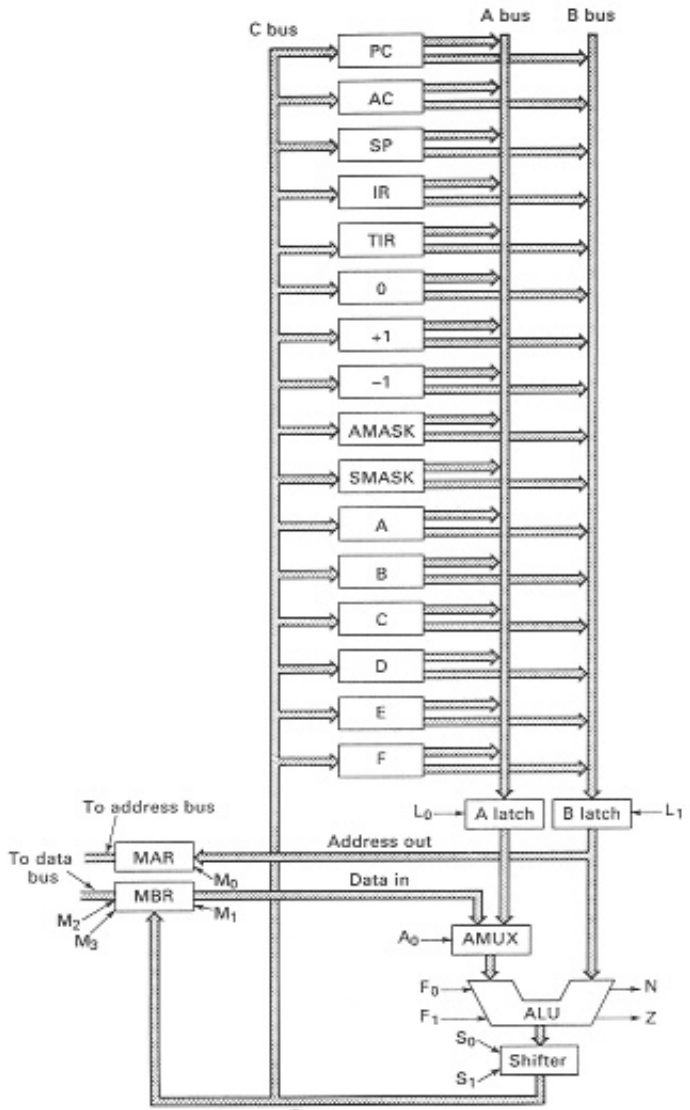
O datapath do processador



Visão geral do processador apresentando os componentes principais e os barramentos A, B, C.

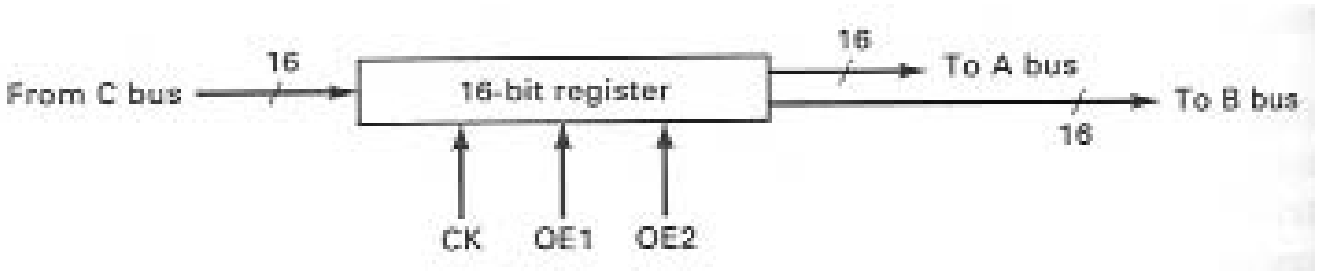
Detalhes a seguir

Sinais de controle



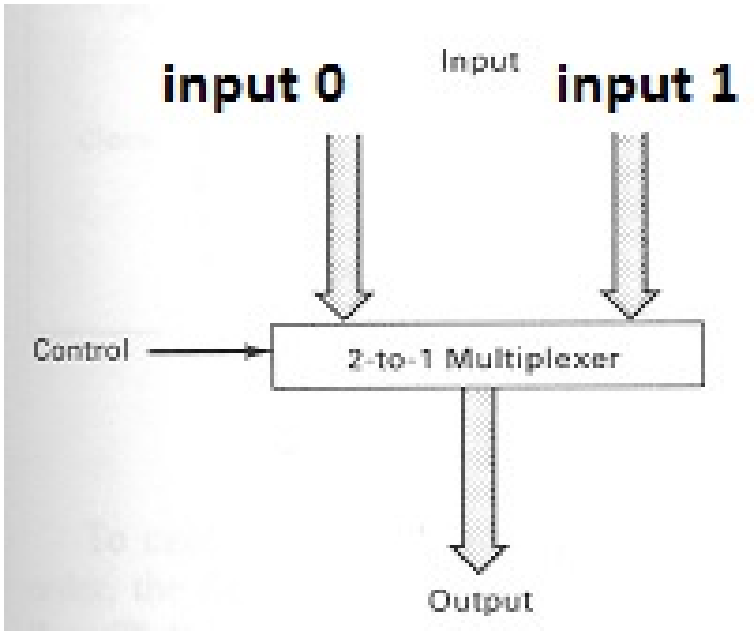
- Em cada ciclo, os componentes da arquitetura MIC são controlados por sinais de controle (veremos a seguir).
- Uma microinstrução basicamente é o conjunto desses sinais de controle.

Sinais de controle de cada registrador



- São 16 registradores no processador.
- Cada um é controlado por 3 sinais de controle:
  - CK=1: valor do bus C é colocado dentro do registrador
  - OE1=1: valor do registrador é colocado no bus A
  - OE2=1: valor do registrador é colocado no bus B

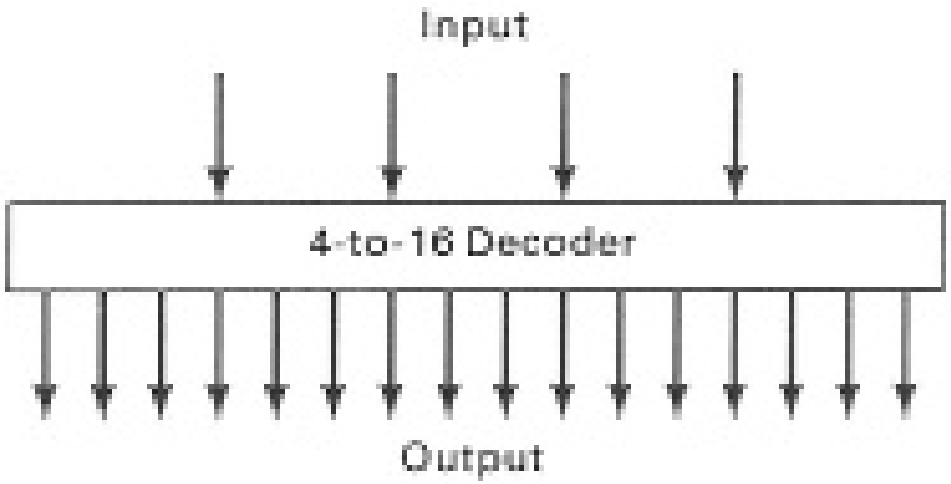
Sinais de controle do multiplexador MUX



O multiplexador MUX é controlado por um sinal (Control):

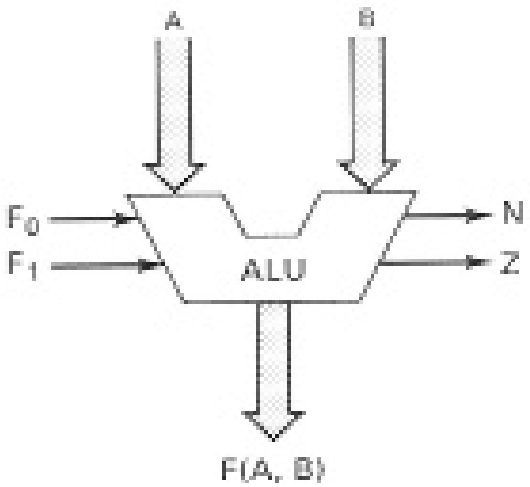
- Control=0: a saída do MUX seleciona o valor do input 0.
- Control=1: a saída do MUX seleciona o valor do input 1.

Decodificador 4-para-16



O decodificador não precisa de sinais de controle.

Sinais de controle da ALU



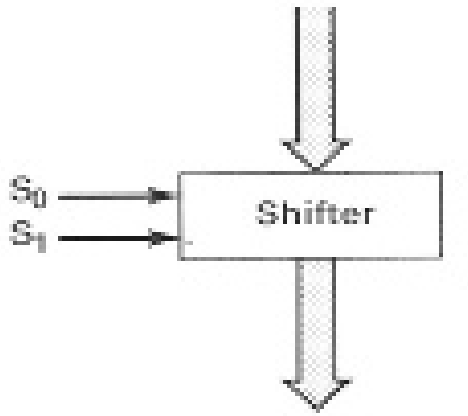
A unidade aritmético-lógica ALU sabe fazer 4 operações. Ela é controlada por dois sinais Fo e F1:

- FoF1 = 00: saída igual a A + B
- FoF1 = 01: saída igual a A and B
- FoF1 = 10: saída igual a A
- FoF1 = 11: saída igual ao complemento de A

ALU produz ainda duas saídas booleanas N e Z:

- N = 1 quando a saída é negativa.
- Z = 1 quando a saída é zero.

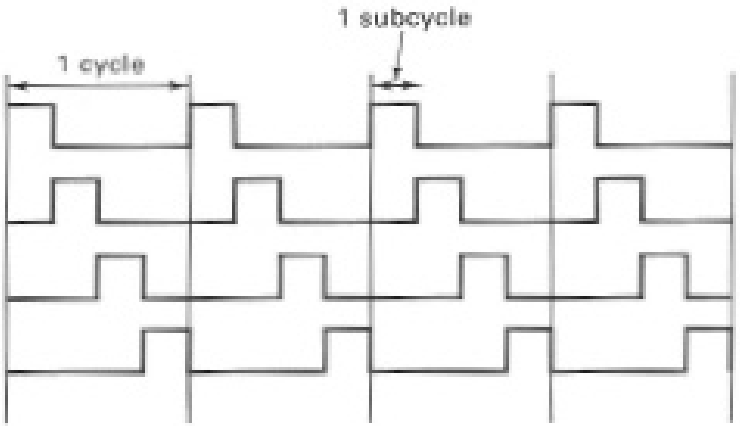
Sinais de controle do Shifter



O Shifter é um deslocador de bits.
Ele é controlada por dois sinais So e S1:

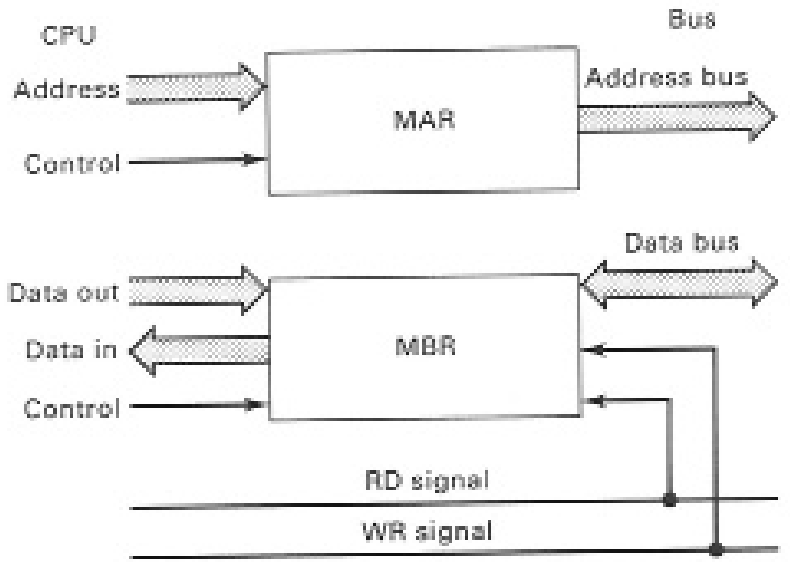
- SoS1 = 00: saída igual à entrada (nada muda)
- SoS1 = 01: desloca entrada de um bit para direita
- SoS1 = 10: desloca entrada de um bit para esquerda
- SoS1 = 11: não usada

Relógio de 4 fases



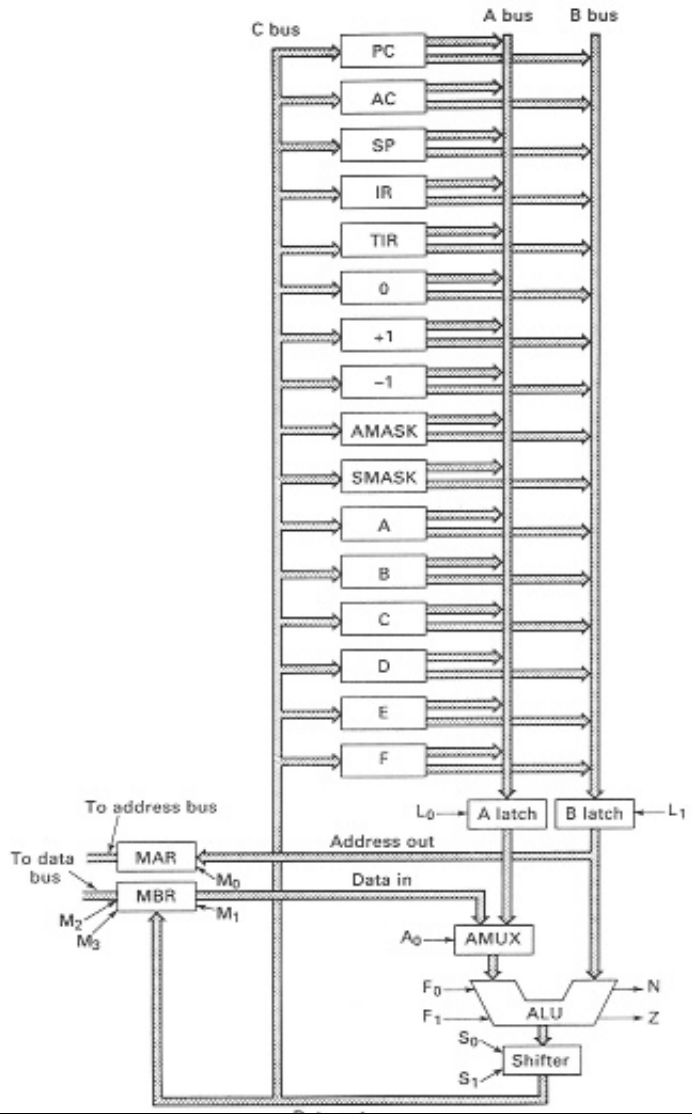
- O ciclo do relógio é dividido em 4 subciclos.
- Cada subciclo apresenta valor alto em um quarto do ciclo.

Sinais de controle de MAR e MBR



- MAR - Memory Address Register:
  Quando control = 1 um endereço é colocado dentro de MAR.
- MBR - Memory Buffer Register: 3 sinais de controle são usados.
  Quando control = 1 um dado é colocado dentro de MBR.
  RD indica leitura e WR indica escrita.

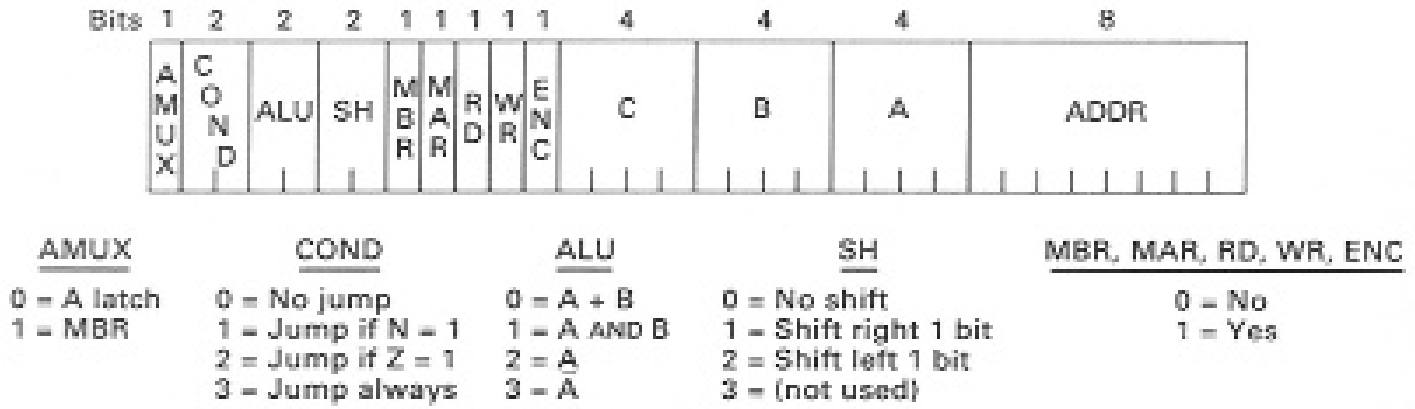
A arquitetura MIC



Formato da microinstrução

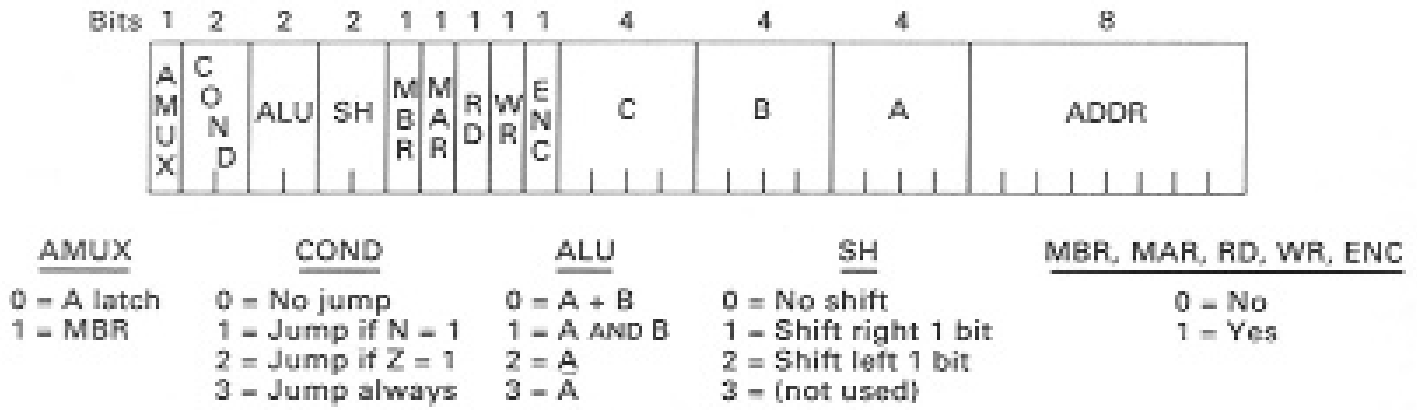
- Uma microinstrução da arquetetura MIC é o conjunto de sinais de controle para o datapath durante um ciclo.
- Um formato possível de uma microinstrução é usar 60 bits:
  - 16 sinais para controlar a carga do barramento A
  - 16 sinais para controlar a carga do barramento B
  - 16 sinais para controlar a carga de registrador pelo barramento C
  - 2 sinais para controlar *A latch* e *B latch* (L_0 e L_1)
  - 2 sinais para controlar a função da ALU (F_0 e F_1)
  - 2 sinais para controlar o shifter (S_0 e S_1)
  - 4 sinais para controlar o MAR e MBR (M_0, M_1, M_2, M_3 - também denominados *MAR*, *MBR*, *RD* e *WR*)
  - 1 sinal para controlar o AMUX (A0)
  - 1 sinal ENC (enable C) para indicar se o resultado calculado deve ser carregado de volta para algum registrador.

Formato da microinstrução



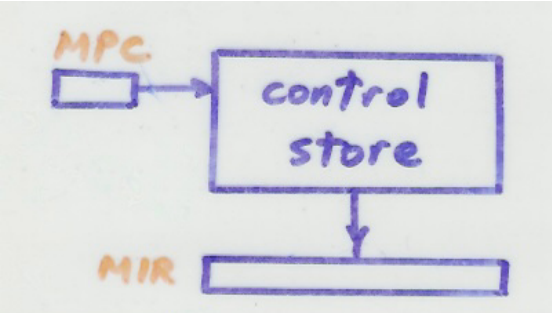
- Ao invés de gastar 16 bits para controlar a carge de valores no barramento A, podemos usar apenas 4 bits codificando um dos 16 valores.
- O mesmo pode ser feito para barramentos B e C.
- Os 2 bits para controlar *A latch* e *B latch* podem ser substituídos pelo sinal de relógio.

Formato da microinstrução



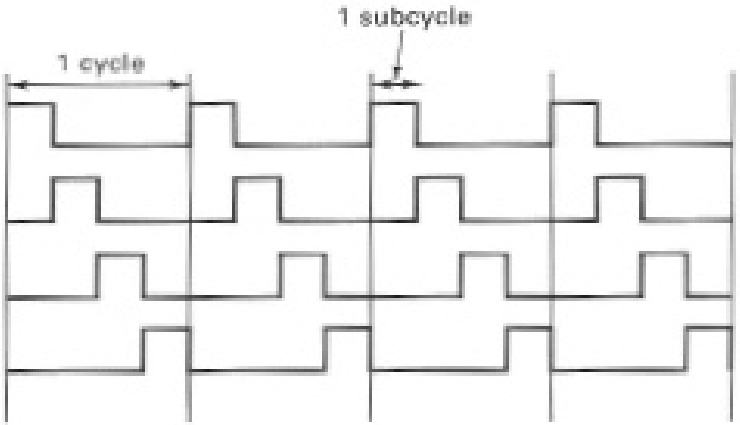
- COND e ADDR: usados para controlar qual a próxima microinstrução a ser executada
  - ADDR: endereço da próxima microinstrução
  - COND: condição para que a próxima microinstrução seja aquela dada por ADDR

Microprograma



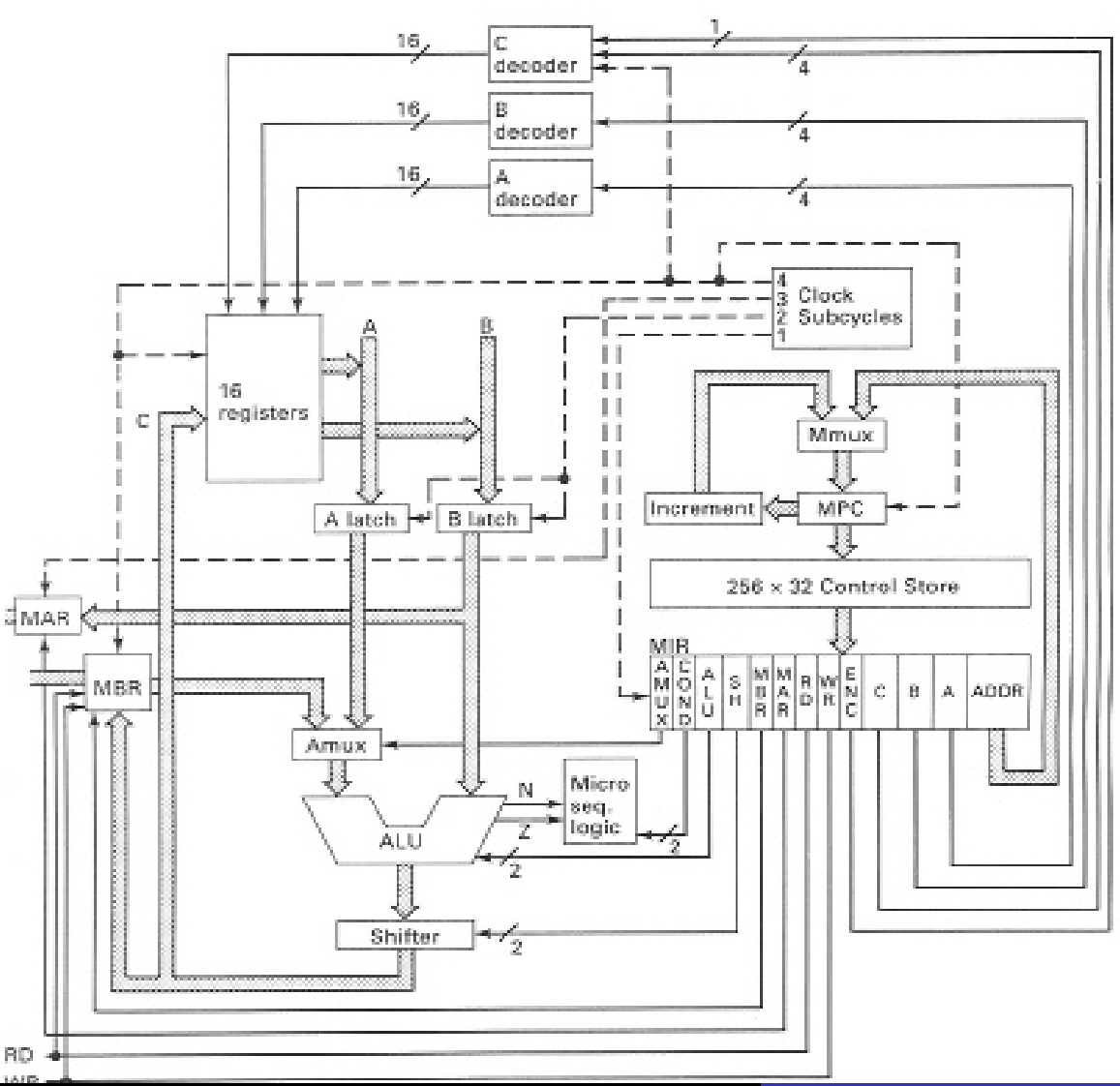
- O microprograma é um conjunto de microinstruções.
- Ele é armazenado numa memória ROM do processador, chamada *control store*.
- Há um MPC (*micro program counter* que aponta para a próxima microinstrução dentro do control store).
- Há um registrador chamado MIR que armazena a microinstrução em execução.

4 subciclos na execução da microinstrução

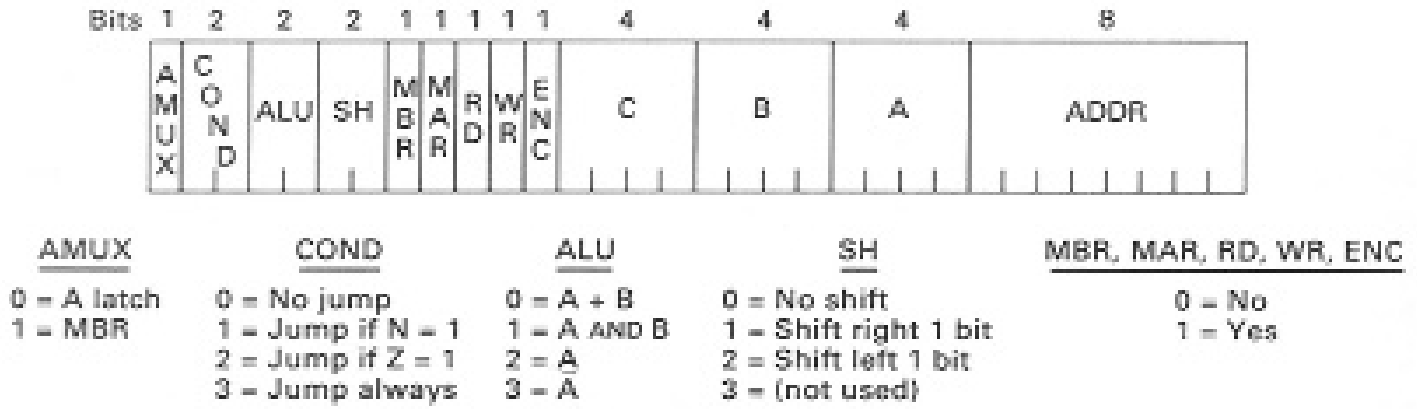


- Subciclo 1: carrega a próxima microinstrução a ser executada num registrador chamado MIR (*micro instruction register*)
- Subciclo 2: coloca valores dos registradores nos barramentos A e B, carregando os *A latch* e *B latch*.
- Subciclo 3: dá o tempo necessário para a ALU e shifter produzirem seu resultado, carregando-o no MAR se for o caso.
- Subciclo 4: armazena o resultado no registrador ou no MBR.

Arquitetura MIC completa com control store

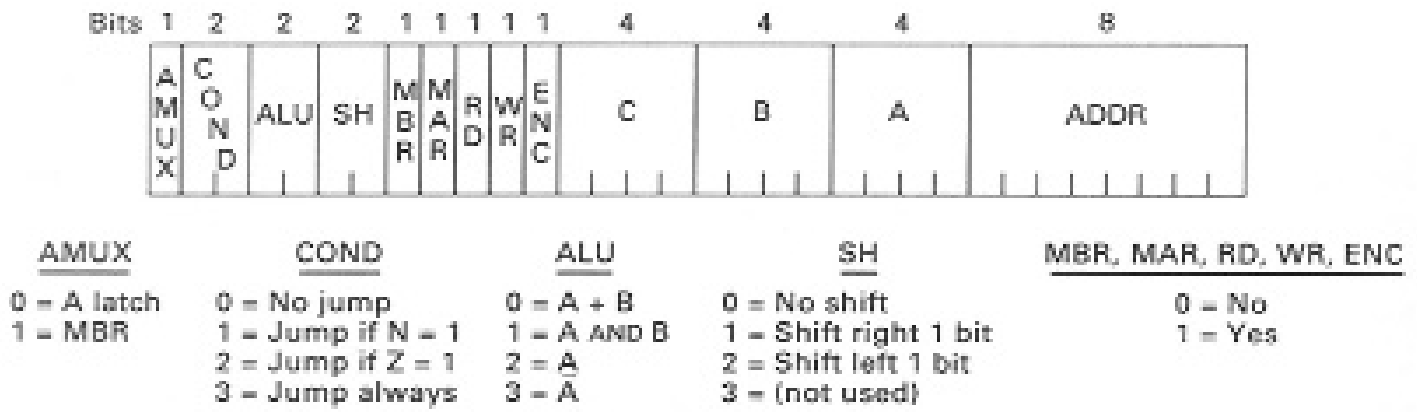


Dificuldade de escrever microinstruções



- Cada microinstrução consta de 32 bits (conforme já visto).
- Esses 32 bits determinam o que deve acontecer num ciclo (4 subciclos).
- Escrever cada microinstrução é uma tarefa árdua (pois lida com o nível muito baixo - zeros e uns).

Micro-assembler



- Micro-assembler (ou micro-montador) facilita a escrita de microinstruções por permitir mnemônicos e símbolos parecidos com um programa de alto nível.
- Na verdade o micro-assembler ainda é baixo nível, no sentido de cada microinstrução em micro-assembler deve corresponder a uma microinstrução de 32 bits.

Exemplo de microinstruções em micro-assembler

- Uma microinstrução em micro-assembler pode ser assim:
  pc := pc + 1
- Ela corresponde a uma microinstrução de 32 bits, onde
  A = 0 (Registrador 0 é PC)
  B = 6 (Registrador 6 contém o número 1)
  C = 0
  ALU = 0 (0 corresponde à operação soma na ALU)
  ENC = 1 (indica que o resultado da ALU deve voltar ao registrador 0)
- Fica mais fácil escrever pc := pc + 1 do que 00000000000100000110000000000000.

Exemplo de microinstruções em micro-assembler

- Uma microinstrução em micro-assembler pode ser assim:
  pc := pc + 1
- Ela corresponde a uma microinstrução de 32 bits, onde
  A = 0 (Registrador 0 é PC)
  B = 6 (Registrador 6 contém o número 1)
  C = 0
  ALU = 0 (0 corresponde à operação soma na ALU)
  ENC = 1 (indica que o resultado da ALU deve voltar ao registrador 0)
- Fica mais fácil escrever pc := pc + 1 do que 00000000000100000110000000000000.

Uso das 4 operações da ALU

Para especificar os 2 bits ALU que controlam a ALU, usamos +, band ou inv. Exemplos:

- pc: := pc + 1
- ac := band (ac, tir)
- tir := inv(tir)

Uso das operações de deslocamento (shifter)

Para especificar os bits SH que controlam o “shifter”, usamos as funções lshift e rshift:

- ac := rshift(ir)
- tir:=lshift(tir+tir)
- Essa microinstrução coloca tir nos barramentos A e B, realiza a adição, desloca o resultado de 1 bit para esquerda, e finalmente guarda o resultado de volta a tir.
- Pergunta: qual o efeito dessa microinstrução? o valor de tir é multiplicado por que valor?

Uso das operações de deslocamento (shifter)

Para especificar os bits SH que controlam o “shifter”, usamos as funções lshift e rshift:

- ac := rshift(ir)
- tir:=lshift(tir+tir)
- Essa microinstrução coloca tir nos barramentos A e B, realiza a adição, desloca o resultado de 1 bit para esquerda, e finalmente guarda o resultado de volta a tir.
- Pergunta: qual o efeito dessa microinstrução? o valor de tir é multiplicado por que valor?

Uso das operações de deslocamento (shifter)

Para especificar os bits SH que controlam o “shifter”, usamos as funções lshift e rshift:

- ac := rshift(ir)
- tir:=lshift(tir+tir)
- Essa microinstrução coloca tir nos barramentos A e B, realiza a adição, desloca o resultado de 1 bit para esquerda, e finalmente guarda o resultado de volta a tir.
- Pergunta: qual o efeito dessa microinstrução? o valor de tir é multiplicado por que valor?

Uso de desvio condicional if

- Desvio incondicional usa o comando goto. Exemplo: goto 12
- Desvios condicionais podem testar N ou Z (saída da ALU negativa ou zero). Exemplos:
- ac:= ac + 1; if z then goto 45
  Note que tudo isso acima pode ser gerado por uma microinstrução de 32 bits.
- Isso é essencial, pois cada microinstruçao em micro-assembler deve corresponder a apenas uma microinstruçao.
- Assim sendo, nada de micro-assember do tipo ac := 7 * ac + sqrt (pc) - cos (tir); if ac < arcsin(sp) then goto 45 :-)

Uso de desvio condicional if

- Desvio incondicional usa o comando goto. Exemplo: goto 12
- Desvios condicionais podem testar N ou Z (saída da ALU negativa ou zero). Exemplos:
- ac:= ac + 1; if z then goto 45
  Note que tudo isso acima pode ser gerado por uma microinstrução de 32 bits.
- Isso é essencial, pois cada microinstruçao em micro-assembler deve corresponder a apenas uma microinstruçao.
- Assim sendo, nada de micro-assember do tipo ac := 7 * ac + sqrt (pc) - cos (tir); if ac < arcsin(sp) then goto 45 :-)

Uso de desvio condicional if

- Desvio incondicional usa o comando goto. Exemplo: goto 12
- Desvios condicionais podem testar N ou Z (saída da ALU negativa ou zero). Exemplos:
- ac:= ac + 1; if z then goto 45
  Note que tudo isso acima pode ser gerado por uma microinstrução de 32 bits.
- Isso é essencial, pois cada microinstruçao em micro-assembler deve corresponder a apenas uma microinstruçao.
- Assim sendo, nada de micro-assember do tipo ac := 7 * ac + sqrt (pc) - cos (tir); if ac < arcsin(sp) then goto 45 :-)

Uso de desvio condicional if

- Desvio incondicional usa o comando goto. Exemplo: goto 12
- Desvios condicionais podem testar N ou Z (saída da ALU negativa ou zero). Exemplos:
- ac:= ac + 1; if z then goto 45
  Note que tudo isso acima pode ser gerado por uma microinstrução de 32 bits.
- Isso é essencial, pois cada microinstruçao em micro-assembler deve corresponder a apenas uma microinstruçao.
- Assim sendo, nada de micro-assember do tipo ac := 7 * ac + sqrt (pc) - cos (tir); if ac < arcsin(sp) then goto 45 :-)

Desvio condicional ao valor de um dado registrador

- Quero escrever uma microinstrução que faz o seguinte. Se ac for zero, então desvia para 47.
- Como fazemos isso?
- Para podermos usar if z then goto 47, o valor de ac deve aparecer na saída da ALU.
- alu:=ac; if z then goto 47
- Isso faz o ac passar pela ALU, apenas para podermos usar o teste if z.

Desvio condicional ao valor de um dado registrador

- Quero escrever uma microinstrução que faz o seguinte. Se ac for zero, então desvia para 47.
- Como fazemos isso?
- Para podermos usar if z then goto 47, o valor de ac deve aparecer na saída da ALU.
- alu:=ac; if z then goto 47
- Isso faz o ac passar pela ALU, apenas para podermos usar o teste if z.

Desvio condicional ao valor de um dado registrador

- Quero escrever uma microinstrução que faz o seguinte. Se ac for zero, então desvia para 47.
- Como fazemos isso?
- Para podermos usar if z then goto 47, o valor de ac deve aparecer na saída da ALU.
- alu:=ac; if z then goto 47
- Isso faz o ac passar pela ALU, apenas para podermos usar o teste if z.

Desvio condicional ao valor de um dado registrador

- Quero escrever uma microinstrução que faz o seguinte. Se ac for zero, então desvia para 47.
- Como fazemos isso?
- Para podermos usar if z then goto 47, o valor de ac deve aparecer na saída da ALU.
- alu:=ac; if z then goto 47
- Isso faz o ac passar pela ALU, apenas para podermos usar o teste if z.

Leitura da memória

- Leitura da memória leva dois ciclos.
- No primeiro ciclo MAR deve receber o endereço a ser lido, e o bit RD ligado. No segundo ciclo, o bit RD deve continuar ligado.
  ciclo 1: mar:=sp; rd

  ciclo 2: rd
- O dado lido fica disponível no MBR no terceiro ciclo.
- Note o desperdício no ciclo 2: uma microinstrução tem 32 bits, dos quais apenas um bit RD está usado.
  Então um bom micrprogramador tentaria aproveitar melhor essa microinstrução, procurando incluir algo que pode ser feito no mesmo ciclo e assim usa melhor a microinstrução.
  ciclo 1: mar:=sp; rd

  ciclo 2: ac:=ac+1; rd

Leitura da memória

- Leitura da memória leva dois ciclos.
- No primeiro ciclo MAR deve receber o endereço a ser lido, e o bit RD ligado. No segundo ciclo, o bit RD deve continuar ligado.
  ciclo 1: mar:=sp; rd

  ciclo 2: rd
- O dado lido fica disponível no MBR no terceiro ciclo.
- Note o desperdício no ciclo 2: uma microinstrução tem 32 bits, dos quais apenas um bit RD está usado.
  Então um bom micrprogramador tentaria aproveitar melhor essa microinstrução, procurando incluir algo que pode ser feito no mesmo ciclo e assim usa melhor a microinstrução.
  ciclo 1: mar:=sp; rd

  ciclo 2: ac:=ac+1; rd

Escrita na memória

- Escrita na memória também leva dois ciclos.
- No primeiro ciclo MBR deve conter o dado a ser escrito e MAR deve conter o endereço em que será escrito o dado. O bit WR deve estar ligado.
- No segundo ciclo o bit WR deve continar ligado.
- Exemplo:
  ciclo 1: mar:=sp; mbr:=ac; wr

  ciclo 2: wr
- Note novamente o desperdício no ciclo 2: dos 32 bits da microinstrução usamos apenas um bit WR. Podemos por exemplo incluir mais coisas no segundo ciclo.
  ciclo 1: mar:=sp; mbr:=ac; wr

  ciclo 2: ac:=ac+1; if z goto 40; wr

Escrita na memória

- Escrita na memória também leva dois ciclos.
- No primeiro ciclo MBR deve conter o dado a ser escrito e MAR deve conter o endereço em que será escrito o dado. O bit WR deve estar ligado.
- No segundo ciclo o bit WR deve continar ligado.
- Exemplo:
  ciclo 1: mar:=sp; mbr:=ac; wr

  ciclo 2: wr
- Note novamente o desperdício no ciclo 2: dos 32 bits da microinstrução usamos apenas um bit WR. Podemos por exemplo incluir mais coisas no segundo ciclo.
  ciclo 1: mar:=sp; mbr:=ac; wr

  ciclo 2: ac:=ac+1; if z goto 40; wr

O microprograma completo na MIC

```
0: mar := pc ; rd ;                                  {main loop}
1: pc := pc + 1; rd ;                                {increment pc}
2: ir := mbr ; if n then goto 28;                    {save, decode mbr}
3: tir := lshift (ir + ir ); if n then goto 19;
4: tir := lshift (tir ); if n then goto 11;          {000x or 001x?}
5: alu := tir ; if n then goto 9;                    {0000 or 0001?}

6: mar := ir ; rd ;                                  {0000 = LODD}
7: rd ;
8: ac := mbr ; goto 0;

9: mar := ir ; mbr := ac ; wr ;                      {0001 = STOD}
10: wr ; goto 0;

11: alu := tir ; if n then goto 15;                  {0010 or 0011?}

12: mar := ir ; rd ;                                 {0010 = ADDD}
13: rd ;
14: ac := mbr + ac ; goto 0;

15: mar := ir ; rd ;                                 {0011 = SUBD}
16: ac := ac + 1; rd ;                               {Note: x - y = x + 1 + not y}
17: a := inv (mbr );
18: ac := ac + a ; goto 0;

19: tir := lshift (tir ); if n then goto 25;         {010x or 011x?}
20: alu := tir ; if n then goto 23;                  {0100 or 0101?}

21: alu := ac ; if n then goto 0;                    {0100 = JPOS}
22: pc := band (ir , amask ); goto 0;                {perform the jump}

23: alu := ac ; if z then goto 22;                   {0101 = JZER}
24: goto 0;                                          {jump failed}

25: alu := tir ; if n then goto 27;                  {0110 or 0111?}
26: pc := band (ir , amask ); goto 0;                {0110 = JUMP}

27: ac := band (ir , amask ); goto 0;                {0111 = LOCO}

28: tir := lshift (ir + ir ); if n then goto 40;     {10xx or 11xx?}
29: tir := lshift (tir ); if n then goto 35;         {100x or 101x?}
30: alu := tir ; if n then goto 33;                  {1000 or 1001?}

31: a := ir + sp ;                                   {1000 = LODL}
32: mar := a ; rd ; goto 7;

33: a := ir + sp ;                                   {1001 = STOL}
34: mar := a ; mbr := ac ; wr ; goto 10;

35: alu := tir ; if n then goto 38;                  {1010 or 1011?}

36: a := ir + sp ;                                   {1010 = ADDL}
37: mar := a ; rd ; goto 13;

38: a := ir + sp ;                                   {1011 = SUBL}
39: mar := a ; rd ; goto 16;
```

Vale a pena examinar as primeiras microinstruções para ver como o microprograma faz a busca de instruções e depois a decodificação da instrução lida.

O microprograma completo na MIC - continuação

```
40: tir := lshift(tir); if n then goto 46;          {110x or 111x?}
41: alu := tir; if n then goto 44;                  {1100 or 1101?}
42: alu := ac; if n then goto 22;                   {1100 = JNEG}
43: goto 0;
44: alu := ac; if z then goto 0;                    {1101 = JNZE}
45: pc := band(ir, amask); goto 0;
46: tir := lshift(tir); if n then goto 50;
47: sp := sp + (-1);                                {1110 = CALL}
48: mar := sp; mbr := pc; wr;
49: pc := band(ir, amask); wr; goto 0;
50: tir := lshift(tir); if n then goto 65;          {1111, examine addr}
51: tir := lshift(tir); if n then goto 59;
52: alu := tir; if n then goto 56;
53: mar := ac; rd;                                  {1111000 = PSHI}
54: sp := sp + (-1); rd;
55: mar := sp; wr; goto 10;
56: mar := sp; sp := sp + 1; rd;                    {1111001 = POPI}
57: rd;
58: mar := ac; wr; goto 10;
59: alu := tir; if n then goto 62;
60: sp := sp + (-1);                                {1111010 = PUSH}
61: mar := sp; mbr := ac; wr; goto 10;
62: mar := sp; sp := sp + 1; rd;                    {1111011 = POP}
63: rd;
64: ac := mbr; goto 0;
65: tir := lshift(tir); if n then goto 73;
66: alu := tir; if n then goto 70;
67: mar := sp; sp := sp + 1; rd;                    {1111100 = RETN}
68: rd;
69: pc := mbr; goto 0;
70: a := ac;                                        {1111101 = SWAP}
71: ac := sp;
72: sp := a; goto 0;
73: alu := tir; if n then goto 76;
74: a := band(ir, smask);                           {1111110 = INSP}
75: sp := sp + a; goto 0;
76: a := band(ir, smask);                           {1111111 = DESP}
77: a := inv(a);
78: a := a + 1; goto 75;
```

O microprograma completo tem apenas 79 microinstruções.

O microprograma completo na MIC - algumas explicações

```
00: mar:=pc; rd
01: pc:=pc+1; rd
02: ir:=mbr; if n then goto 28
03: tir:=lshift(ir+ir); if n then goto 19
04: tir:=lshift(tir); if n then goto 11
05: alu:=tir; if n then goto 9
06: mar:=ir; rd
07: rd
08: ac:=mbr; goto 0
09: mar:=ir; mbr:=ac; wr
10: wr; goto 0
11: alu:=tir; if n then goto 15
12: mar:ir; rd
13: rd
14: ac:=mbr + ac; goto 0
15: mar:=ir; rd
16: ac:=ac+1; rd
.
.
```

Como escrever microprograma eficiente

- Uma microinstrução tem 32 bits que comandam a ação do processador durante um ciclo.
- É importante explorarmos, se possível, todos esses bits disponíveis numa mesma microinstrução, ao invés de dividir algo que poderia ser feito em um ciclo para serem feitos em dois ciclos ou mais.
- Leitura (rd) leva dois ciclos. Assim rd deve aparecer em duas microinstruções seguidas. O mesmo para escrita (wr). Então é importante aproveitar a microinstrução e não definir uma microinstrução apenas com rd ou apenas com wr.
- Veremos um exemplo a seguir de como escrever um microprograma eficiente.

Exemplo de como escrever um microprograma eficiente

- Suponha a criaçã de uma nova instrução de máquina (nível convencional) chamada NOVA que faz o seguinte:

  Escreve o valor 0 na memória endereçada por SP
  Faz AC ficar igual a 4 vezes o valor de SP
  Soma TIR em AC e se o valor da soma ficar negativa
  entao faz AC igual a 0, senão faz AC igual a 1
  No final o controle deve voltar a posição 0.

- Suponha que a instrução NOVA já está lida e encontra-se no IR. Suponha ainda que já foi feita a decodificação e sabe-se que se trata da instrução NOVA.
- Vamos escrever, em micro-assembler, o trecho das microinstruções que implementa NOVA. Suponha que o início desse trecho é na linha 101.

Solução: microprograma para NOVA

Escreve o valor 0 na memória endereçada por SP
Faz AC ficar igual a 4 vezes o valor de SP
Soma TIR em AC e se o valor da soma ficar negativa
entao faz AC igual a 0, senão faz AC igual a 1
No final o controle deve voltar a posição 0.

Ineficiente:

```
101: mar:=sp;
102: mbr:=0; wr
103: wr
104: ac:=(sp+sp)
105: ac:=ac+ac
106: ac:=ac+tir; if n goto 109
107: ac:=1
108: goto 0
109: ac:=0
110: goto 0
```

Eficiente:

```
101: mar:=sp; mbr:=0; wr
102: ac:=lshift(sp+sp); wr
103: ac:=ac+tir; if n goto 105
104: ac:=1; goto 0
105: ac:=0; goto 0
```

Solução: microprograma para NOVA

Escreve o valor 0 na memória endereçada por SP
Faz AC ficar igual a 4 vezes o valor de SP
Soma TIR em AC e se o valor da soma ficar negativa
entao faz AC igual a 0, senão faz AC igual a 1
No final o controle deve voltar a posição 0.

Ineficiente:

101: mar:=sp;
102: mbr:=0; wr
103: wr
104: ac:=(sp+sp)
105: ac:=ac+ac
106: ac:=ac+tir; if n goto 109
107: ac:=1
108: goto 0
109: ac:=0
110: goto 0

Eficiente:

101: mar:=sp; mbr:=0; wr
102: ac:=lshift(sp+sp); wr
103: ac:=ac+tir; if n goto 105
104: ac:=1; goto 0
105: ac:=0; goto 0

Outro exemplo de como escrever um microprograma eficiente

Como escrever microprogramas eficientes.
Exercício em classe.

Microinstruções horizontais e verticais

Uma microinstrução especifica os sinais de controle necessários para controlar a microarquitetura.

- Microinstrução horizontal:
  - Todos os sinais necessários estão colocados na mesma microinstrução, sem nenhuma codificação.
  - O “control store” contém um pequeno número de microinstruções compridas formadas com muitos campos, daí o nome horizontal.
- Microinstrução vertical:
  - A microinstrução contém poucos campos, altamente codificados.
  - Mais de uma microinstrução podem ser necessárias para especificar todos os sinais necessários.
  - O “control store”contém em geral um grande número de microinstruções curtas, daí o nome vertical.

O que pode se concluir de microprogramação?

O que pode se concluir de microprogramação?

- Ela é bem chata :-)
- Mas falando sério: microprogramação é uma técnica poderosa que permite implementar insruções complexas de um repertório grande de instruções, num hardware simples.

O que pode se concluir de microprogramação?

O que pode se concluir de microprogramação?

- Ela é bem chata :-)
- Mas falando sério: microprogramação é uma técnica poderosa que permite implementar insruções complexas de um repertório grande de instruções, num hardware simples.

O que pode se concluir de microprogramação?

O que pode se concluir de microprogramação?

- Ela é bem chata :-)
- Mas falando sério: microprogramação é uma técnica poderosa que permite implementar insruções complexas de um repertório grande de instruções, num hardware simples.